Computer-aided design of a generic robot controller handling reactivity and real-time control issues

Daniel Simon, Bernard Espiau, Eduardo Castillo, Konstantin Kapellos

Rapports de Recherche

N°1801

Programme 4

Robotique, Image et Vision

COMPUTER-AIDED DESIGN OF A GENERIC ROBOT CONTROLLER HANDLING REACTIVITY AND REAL-TIME CONTROL ISSUES

Daniel SIMON
Bernard ESPIAU
Eduardo CASTILLO
Konstantinos KAPELLOS

Novembre 1992

UNITÉ DE RECHERCHE
INRIA-SOPHIA ANTIPOLIS

Institut National
de Recherche
en Informatique
et en Automatique

2004 route des Lucioles
B.P. 93
06902 Sophia-Antipolis
France

Computer-Aided Design of a Generic Robot Controller Handling Reactivity and Real-Time Control Issues

Conception Assistée par Ordinateur d'un Contrôleur de Robots Générique Associant Réactivité et Commande en Temps Réel

Daniel SIMON †, Bernard ESPIAU †, Eduardo CASTILLO †
Konstantinos KAPELLOS ‡

(†) INRIA
2004 Route des Lucioles
BP 109 06902 Sophia-Antipolis France

(‡) ISIA/ENSMP
rue Claude Daunesse
06565 Valbonne Cedex - France

Programme 4
Robotique, Image et Vision

Rapport de Recherche Inria no 1801
Novembre 1992

Abstract

This paper describes an original system for the computer-aided design of robot controllers, the ORCCAD system. Accessed by three different user levels (system, control and application), it proposes a coherent approach from high level specification down to implementation, and offers several tools for design, display and test.

Following a critical study of the main architectures reported through the literature, the paper presents the basic principles and underlying concepts of ORCCAD. The main considered entity is the *robot task*, elementary control action associated with a local behavior controlled by a set of observers and modeled by a finite state automaton. Heart of the system, this robot task, which is based on a generic control scheme, is then described in details. It is made of a set of real-time tasks which communicate, called *module tasks*, the full definition of which requires the specification of temporal and synchronization features. The module task which handles the behavior of the robot task is described using the synchronous language ESTEREL, and its code is automatically generated. The application level, presented in the next section, is defined as a set of synchronized robot tasks. Let us emphasize that ESTEREL is again the language used to describe it.

Then, the paper gives some details on the dialogue and verification tools integrated within ORCCAD: a graphic Human-Machine Interface (HMI) and a simulation system. This last includes two subsystems: a first simulator allows to validate control schemes taking into account discretization aspects, while the second is aimed to the description and the test of a multiprocessor implementation. The simulation code is automatically generated from the HMI. Finally, two examples are discussed. The first one consists in the specification and the test of a few versions of a joint control for a rigid robot; the other one is a mission specification for an autonomous underwater vehicle. An appendix and 40 references end the paper. [1]

Résumé

Ce rapport de recherche présente un système original d'aide à la conception de contrôleurs de robots, le système ORCCAD. Accessible à trois niveaux différents d'utilisateurs (système, commande, application), il propose une approche cohérente, depuis la spécification haut niveau jusqu'à l'implémentation, et offre un ensemble d'outils de conception, de visualisation et de simulation.

Après une évocation critique des principales architectures décrites dans la littérature, le rapport présente les principes généraux et les concepts sous-jacents au système ORCCAD. L'entité principale considérée est la *tâche robot*, action élémentaire de commande associée à un comportement local contrôlé par des observateurs et représenté par un automate d'états fini. Coeur du système, cette tâche robot basée sur un schéma de commande générique fait ensuite l'objet d'une description détaillée. Elle est constituée de tâches temps-réel communicantes, les *tâches modules*, dont la définition complète nécessite la spécification de contraintes temporelles et de primitives de synchronisation. La tâche module gérant le comportement est quant à elle décrite à l'aide du langage synchrone ESTEREL, dont le code est généré automatiquement. Le niveau application, objet du chapitre suivant est défini comme un ensemble de tâches robot synchronisées. Soulignons qu'à nouveau le langage ESTEREL est utilisé pour la description des relations temporelles et logiques entre les tâches robot concernées.

Le rapport présente ensuite les outils de dialogue et de vérification intégrés dans ORCCAD: une interface homme-machine graphique orienté objet et un sytème de simulation. Celui-ci comporte deux aspects: un premier simulateur permet de valider les schémas de commande prenant en compte les aspects liés à la discrétisation, et un second permet de décrire et tester une implémentation multiprocesseurs. L'interface homme-machine génère automatiquement le code destiné au premier simulateur.

Le rapport s'achève sur la présentation de deux exemples. L'un consiste en la description et le test de diverses versions d'une commande articulaire de robot rigide, l'autre est la spécification de la mission d'un robot autonome sous-marin. Une annexe et quarante références bibliographiques complètent le document.

[0]Part of this work is made in cooperation with Aleph Technologies, Grenoble, France with support from Ministère de la Recherche et de l'Espace under grant #91 S 0097.

Contents

1 Introduction

For the last ten years, many kinds of software architectures for the intelligent control of robotic systems have been designed. Different techniques issued from cognitive sciences as well as automatic control domains are there used. Other important aspects of these software architectures are real-time algorithmic and implementation issues. Although a complete robot control architecture includes various subsystems, developments generally focused only on few of them. Some approaches concentrate on system design, i.e. functional decomposition, while others mainly investigate implementation issues. Ad-hoc tools are then to be used in order to complete the system design, while other important aspects like verification of specifications and mapping of functional tasks on a hardware architecture fall in the gap ([27]). Let us now examine the main approaches to this problem of robot controller design.

Some existing implementation-oriented approaches rely on dedicated real-time operating systems like Chimera [40], using for example fast interprocess communication tools ([33]). Powerful distributed machines have been built ([29]), or layered software libraries have been designed ([5]). However, although these approaches are efficient at the lowest level, they do not consider other levels in a coherent way and they generally provide the end-user neither with friendly interfaces nor with any programming methodology.

Hierarchical software architectures like NASREM ([1]) have been proposed in order to enforce modularity and software development methodology. Within this approach, a measurement data path flows from sensors to the highest levels, and a control data path is fed back to actuators. Such a structure is very rigid, since all "intelligent" actions, like planning using sensors, can only be defined at the high level, while communications between low or intermediary ones are limited or forbidden. This is a major drawback to the building of, for example, real-time efficient sensor-based control.

High level programming of robots traditionally falls in the field of Knowledge-Based Systems and mainly deals with planning and reasoning. For example, learning, on-line planning, or world understanding are common investigation areas. Generally, purely cognitive approaches resolutely ignore the basics of mechanics, control and real time computing. This results in techniques which are not time-efficient and have difficulties to take into account some of the uncertainties issued from the real world where robots operate. As a counterpart to such classical A.I. approaches, the application of behaviorism to robots was proposed by Brooks ([12]).

Actually, behaviorism was firstly introduced as a model of animal and human psychology ([41]). Within this theory, the activity of living beings may be modeled by a set of behaviors. Each elementary behavior is characterized by a reaction to a set of stimuli coming from the environment. The global behavior is considered as a simple juxtaposition of elementary behaviors using low level inhibition mechanisms, without coordination coming from an upper mind level. This very mechanistic point of view of mental processes, negating in fact consciousness, was not confirmed by experimentation and was fought by both psychologists and philosophers ([32]).

When applied to robots, behaviorism led to layered control based on the so-called subsumption architecture. The layered control system is organized as communicating software modules corresponding to *levels of competence*. New behaviors are added whenever a more complex behavior is required. Conflicts due to competing reactions are solved by inhibiting the outputs of the lowest priority modules.

Thus, a top-level supervisor does not exist: control of the overall behavior is distributed along a set of modules and becomes difficult to analyze and validate, as soon as the set of elementary behaviors increases. Although small insect-like robots have been built following this approach, inappropriate or unexpected responses may be generated, as mentioned in ([6]).

Another drawback comes from the subsumption architecture. Its simplicity implies a small system overhead. However, running permanently *all* the behaviors even when not necessary, and

simply adding new processors when new behaviors are required, is clearly not cost-effective.

Nevertheless, this purely reactive approach made a fruitful breach in the classical A.I. point of view and received further adaptations. Let us cite for example the "State Configured Layered Control" ([6]) where subsets of behaviors are grouped into states under control of a state transition diagram. Using such a supervision level allows a more comprehensive control of applications and improves real-time efficiency.

Finally, hybrid architectures, which gather features of previously mentioned approaches, are now emerging. The Rational Behavior Model ([13]) is an example of such an architecture: a mission is specified using a rule-based strategic level and concurrent repetitive behaviors are implemented in a tactical level. Their outputs are used by servo-loops at the execution level. Other interesting examples of hybrid approaches can be found in [19] and [9].

The work described in this paper falls in this last category and is based upon the main following considerations:

- most actions to be performed by robots can be stated as control problems which can be efficiently solved in real-time by using adequate feedback control loops. We believe that control theory should be used as far as possible to specify complex actions. In this framework, the *Task-Function* approach, ([37]) specifically developped for robotic systems which may involve redundant and sensor-based control tasks, is the one used here;
- a robotic system should ideally be accessible to users with different competences. In particular, the end-user of the system is generally neither a control expert nor a computer scientist. He must be provided with high level functions, allowing him to concentrate on application specification and verification rather than on low level programming tricks. Besides, the control scientist must be provided with design and programming tools to help him in designing actions. In the approach we propose, every type of user is provided with a specific access to the system.
- since the overall performances of the system finally relies on the existence of efficient real-time mechanisms at the execution level, particular attention is paid to their specification and their verification.
- robotic control often requires the use of complex algorithms, the programming and test of which take a long time. This is why Object-Oriented Design and Programming are used here in order to improve software reliability and reusability. Automatic code generation is also used as much as possible.

This paper is organized as follows: the next section gives an overview of the *Open Robot Controller Computer Aided Design* system (ORCCAD). In section 3, the Robot-Task concept is analyzed in depth. Section 4 is devoted to the application level accessible to the end-user. Sections 5 and 6 describe tools designed during the project, a Human-Machine Interface and Simulation Software. Two examples are provided in section 7. Guidelines for the future are drawn in conclusion.

2 The ORCCAD Concept

This section gives an overview of the main features of the ORCCAD system. Most of them will be detailed later

2.1 General Approach

A robotic process may be defined as a set of robot actions organized such as to realize a given application, like an assembly operation or the mission control of a mobile robot. The design of

a robotic process needs expertise in several domains: knowledge in mechanics is often required to properly define the task to perform, automatic control theory is involved in the design of control laws and tools from computer science are needed to produce efficient runtime software. Although the potential performance of robotics systems have increased very quickly, for example by using multiprocessor controllers, the development of an efficient design methodology taking into account these different issues did not generally follow.

The reason is that the achievement of the tasks involved in a robotic process via a distributed control architecture raises in fact difficult problems. Several keypoints in the design and the implementation of an application can thus be identified. In a first step, it is necessary to define all the involved elementary tasks. These are for example the atomic entities handled by a planning system. For each of them, issues from automatic control and implementation aspects have to be considered:

- definition of a regulation problem which may be significantly related to the elementary task;
- choice of a suitable control law;
- selection of the events liable to be considered during the task execution and definition of the associated reactions;
- decomposition of the task into real-time subtasks, the timing and synchronization parameters of them being to be determined.

Finally, all the defined real-time subtasks should be mapped on a target architecture.

A lot of degrees of freedom are therefore given to the control designer in order to match an end-user specification. The aim of the ORCCAD system is to help the user to exploit these degrees of freedom in the more efficient way. The associated controller itself is naturally *open*, since access to every control level is allowed to qualified users: the *application* layer is accessed by the *end-user* of the system, the *control* layer is programmed by an *automatic control* expert and the *system* layer, which contains basic tools such as a real time operating system and a set of synchronization mechanisms, is the charge of a system engineer.

The structure of the hardware and software resources used by such an Open Robot Controller to achieve real time control of robotic systems has been studied in a previous work [10, 16]. In order to precise the concepts which underly all the ORCCAD approach, let us recall the basic issues defined in these references. Precise definitions and further details on these entities will be given in the following.

- a *Robotic System* is a set of cooperating devices (such as robots and sensors) associated with a control system.
- a *Robot Controller* is the set of computational resources (hardware and software) involved in the on-line control of a robotic system.
- an *Application* is a sequence of actions performed by the robotic system in order to reach a goal specified by an end-user. Generally, at a certain level, an application can be modeled by an automaton.
- a *Robot-Task* (called RT in the following) is a multitasks program representing robotic actions. It merges algorithmical aspects (control law) and logical aspects (local behavior) and constitutes the elementary task previously evoked.
- a *Module-Task* (called MT in the following) is a real-time task. Therefore, a RT is made of a connected set of MTs.

The ORCCAD system is a set of CAD tools allowing to design, test and implement applications, Robot-Tasks and Module-Tasks as defined above. In fact, ORCCAD allows dedicated users to address all the keypoints of the design process previously given, the central issue being the Robot-Task. The description of Robot-Tasks, down to the specification of its temporal parameters, is made through a specific Human-Machine Interface, while verification and simulation tools are available at the implementation level. These simulation aspects are described in section 6.

Because of the particular importance of the Robot-Task, we now present its *design methodology*, as it appears in the ORCCAD philosophy. The RT structure itself will be further detailed in section 3.

2.2 A Methodology for the Design of Robot-Tasks

The creation process of a RT starts from an end-user specification and is developed step by step towards the generation of real-time tasks to be downloaded. Figure 1 illustrates the involved successive steps. Let us focus on the main items.

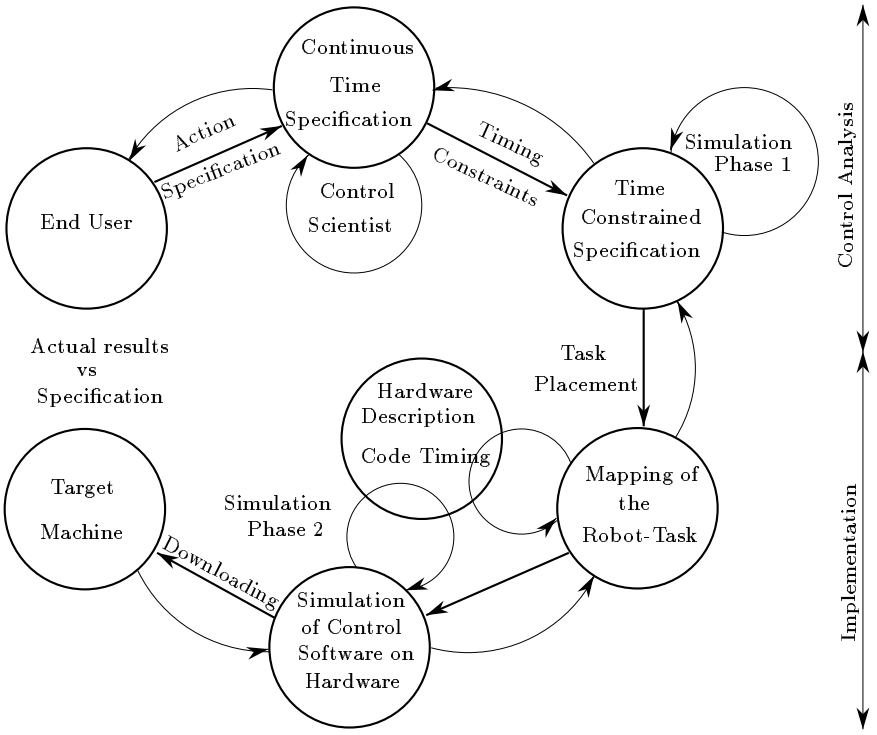

Figure 1: Robot-task creation process

2.2.1 Step 1: Specification of a Robot-Task

A new RT specification is provided by the end-user whenever a suitable RT cannot be found in the ORCCAD library. This description is done in a natural-like language and includes:

- a description of the robotics system to be used: mechanical characteristics of robot arms, characteristics of available sensors ...
- a description of the task, specifying in particular:

- the action to be performed during its execution, including the associated performances indices to be monitored (for example, the maximum allowed tracking error);
- the events to be taken into account during the task execution and the actions to be undertaken upon reception of the generated exceptions
- the conditions authorizing the task to start and the ones generated at its completion.

From this description, the automatic control expert can then propose symbolic descriptions in continuous time of the control laws liable to realize the specified action. This formal specification of control laws is given in term of block-diagrams and/or equations. Constraints on some parameters may also be associated with the mathematical description.

As previously evoked, a systematic way to design control laws from the task specification is the "Task-Function approach". Owing to this approach, and at least for rigid robot arms, a specific control law may be derived from a generic control scheme. Inside this general structure, the RT designer can select adequate items among a large choice of models and algorithms. The general control structure is organized as a set of class hierarchies.

2.2.2 Step 2: Time Constrained Specification

The control algorithm may now be split into smaller communicating tasks, the so-called Module-Tasks. Such a decomposition is in fact close to a block-diagram description of the control law.

The MTs communicate via message passing, using synchronization mechanisms and ports described in section 3.2.1.

A MT owns temporal properties, like its computing duration and its activation period. These properties are attributes, which are added at this step to the continuous time specification of the control law. The values of these temporal properties may be set either using a-priori knowledge, or from scratch for pure control study purpose.

The set of MTs, of their temporal properties and of the chosen synchronization scheme defines the Time Constrained Specification (TCS) of the Robot-Task. Many properties would now be checked at the TCS level before continuing towards implementation. Unfortunately, from the control point of view, the available analysis tools do not give quantitative results on such non-linear systems driven by sampled, distributed and multi-clocked control loops. Besides, liveness properties must also be checked on the synchronization scheme to avoid, at least, deadlocks. These properties rely on the choice of the synchronizations between the MTs and the interleaving of communication instants. Although using Timed Petri Nets and Synchronous Languages description are currently investigated for this purpose, formal tools do not exist yet to check the correctness of the temporal scheme of RTs. So far, the unique way remains simulation for which specific tools have been developed (see section 6).

The simulation phase associated with this step is the first one proposed in ORCCAD. It takes into account not only the algorithmical and temporal attributes of the TCS, but also the physical behavior of the controlled system, modeled as a set of differential equations. Thus, according to the constraints defined by the end-user, it may help the RT designer to tune performance-related parameters, such as value of gains, sampling period of control loops and models to be used. Finally, it may also give a first rough scaling for the target architecture.

2.2.3 Step 3: Implementation of the Robot-Task

Once the first simulation phase allowed the designer to conclude that an implementation was feasible, the MTs have to be distributed on a architecture, generally multiprocessors.

It is well known that optimal task scheduling on a distributed architecture is a NP-hard problem. Although many algorithms and heuristics have been developed, only few of them take into account real time constraints like the respect of deadlines and of sampling periods. In

that framework, the facilities offered by ORCCAD simply consist in allowing to easily specify a distribution and validate it by a dedicated simulation step. New attributes are thus added to the MTs in this step of task placement. These attributes are related to the operating system calls used to instantiate and activate real time tasks.

It is known that multitask programs running on multiprocessors architecture are difficult to debug. Moreover, from a safety point of view, it is better to perform the first tests on a workstation than with the physical system itself. For that reasons the second simulation phase of ORCCAD , close to the hardware, is of some help. It uses the SIMPARC simulator described in section 6, before downloading the tasks on the target machine.

3 The Robot-Task in Depth

The Robot-Task is a keystone concept in the ORCCAD framework: it is the minimal granule to be handled by the *end user* at the application level, while it is the object of maximum complexity to be considered by a *control designer*. It characterizes in a structured way the set of elements allowing the controller to actually work: control scheme in closed loop, temporal features related to implementation, management of associated events (internal and external). It is defined in a formal way as follows:

A Robot-Task is the entire parametrized specification of:

- *an elementary servo-control task, i.e. the activation of a control scheme structurally invariant along the task duration;*
- *a logical behavior associated with a set of signals liable to occur previously to and during the task execution.*

It should be emphasized that a Robot-Task is dedicated to work with a fixed set of physical resources. Note also that a Robot-Task is of given duration, since:

The Robot-Task duration is defined as the one, T, of the elementary servoing task, a-priori determined under the assumption of proper working, and starting at the activation of the servoing task.

An Object-Oriented approach was chosen for modeling the Robot-Task. A given Robot-Task is then fully specified by the instantiation of the concerned objects. As seen in the previous section, this requires the definition of the elementary servoing task: in a first step, the formal specification in *continuous* time has to be established. That characterizes the task from the automatic control point of view. Then, it has to be extended to take into account implementation issues: discretization, variable quantization, delays, computation times, periods... Finally, starting, stopping, killing the task and controlling its evolution require to define adequate signals and to build an automaton managing the overall behavior.

Let us now describe the different aspects successively.

3.1 Continuous Time Control Specification

3.1.1 Introduction

In its present implementation, the system is dedicated to the design of control schemes and to the definition of tasks for a particular class of mechanical systems: the rigid robot manipulators. Nevertheless, we may address a large variety of mechanical structures owing to the connection of ORCCAD with *ACT*, a robotics-oriented CAD system ([30]). Moreover, considered tasks include

complex ones which use various exteroceptive sensors: force, range, proximity, vision. The class of potential applications is therefore very wide. Let us now examine, briefly since it is not the aim of this paper, the proposed control scheme. Its theoretical study as well as the discussion of various examples of applications may be found in [37] and [21].

The basic principle consists in separating, in the *design* step, the specification of the task to be performed from the determination of the low-level control law, while properly melting both aspects in the *achievement* of the global control scheme. The first aspect consists of expressing the user's objective under the form of an adequate output function $e(q,t)$, called *task function*. The second consists in choosing the set of models to be implemented and in tuning the parameters depending on required performances, computing issues and considered mechanical system. In that way, changing a robot for a given task, specifying various tasks for a given robot or refining a control law for a given couple {task, robot} are handled within a single framework.

3.1.2 A General Control Scheme

Preliminary, let us point out that the study of this control scheme is the unique topic of [37]. This is why we will only give here a simple view of the problem.
The dynamic (state) equation of a $n-$jointed robot, with joint coordinates denoted as q is:

$$\Gamma = M(q)\ddot{q} + N\left(q,\dot{q},t\right) \tag{1}$$

where Γ is the $n-$vector of joint actuator torques, M is the kinetics energy matrix, and N gathers all other dynamical terms.

Let us now consider the n-dimensional C^2 task-function, $e(q,t)$, to be regulated to zero during the time interval $[0,T]$, starting from an initial position q_0. The simplest example of such a function corresponds to trajectory tracking in joint space: $e(q,t) = q - q_d(t)$. Many more interesting choices are offered to the user in the ORCCAD system (see Section 3.3.1).

Once it has been verified that the problem is well-stated, i.e that it exists a C^2 solution to equation $e(q,t) = 0$ and that some regularity conditions of the *task-jacobian* $\frac{\partial e}{\partial q}(q,t)$ are satisfied, it remains to find the control Γ. By writing:

$$\dot{e} = \frac{\partial e}{\partial q}(q,t)\dot{q} + \frac{\partial e}{\partial t}(q,t) \tag{2}$$

$$\ddot{e} = \frac{\partial e}{\partial q}(q,t)\ddot{q} + f(q,\dot{q},t), \tag{3}$$

it may be seen that a control which ideally decouples and ensures an asymptotically stable linear behavior of the error e is ([37]):

$$\Gamma = M\left(\frac{\partial e}{\partial q}\right)^{-1}\left[-kG\left(\mu De + \dot{e}\right)\right] + N - M\left(\frac{\partial e}{\partial q}\right)^{-1} f \tag{4}$$

where G and D are positive matrices and k and μ positive scalars.

The ideal control scheme (4) is based on a perfect knowledge of all its components. In ORCCAD, this control is implemented under the more general form:

$$\Gamma = -k\hat{M}\left(\widehat{\frac{\partial e}{\partial q}}\right)^{-1} G\left(\mu De + \widehat{\frac{\partial e}{\partial q}}\dot{q} + \widehat{\frac{\partial e}{\partial t}}\right) + \hat{N} - \hat{M}\left(\widehat{\frac{\partial e}{\partial q}}\right)^{-1}\hat{f} \tag{5}$$

where, finally:

- $k(.)$ is a positive scalar gain, which is allowed to be a nonlinear function of e, q and $\dot{q}$. It has the intuitive meaning of a velocity gain, while the positive scalar μ may be interpreted somewhere as the ratio between velocity and position gains.

- G and D are constant positive adjusting matrices, generally diagonal.

- $\hat{M}(q,t)$ is a $n \times n$ symmetric positive matrix chosen as a model of the system kinetics energy matrix.

- $\hat{N}(q,\dot{q},t)$ is a $n-$ dimensional vector function chosen as a model of the sum of the terms of gravity, friction, Coriolis and centrifugal forces.

- $\widehat{\frac{\partial e}{\partial q}}(q,t)$ is a $n-$ dimensional vector function chosen as a model of the task-function jacobian matrix, while $\widehat{\frac{\partial e}{\partial q}}^{-1}(q,t)$ is the model chosen for its inverse.

- $\widehat{\frac{\partial e}{\partial t}}$ is a $n-$ dimensional vector function chosen as a model of the time partial derivative of the task function.

- $f(q,\dot{q},t)$ is a $n-$ dimensional vector function chosen as a model of the terms corresponding to the second-order differentiation of e (cf eq. 3).

- equation 5 may also include an error integral term.

In general, constraints on actuators and joint limits also exist:

- $max({\Gamma_i}^{min}, -\alpha_i - \beta_i\dot{q}_i) \leq \Gamma_i \leq min({\Gamma_i}^{max}, \alpha_i - \beta_i\dot{q}_i) \quad \forall i \in [1,n]$
  where Γ_i is the ith scalar control torque, α_i and β_i are constant and ${\Gamma_i}^{min}$ and ${\Gamma_i}^{max}$ are torque limits;

- ${q_i}^{min} \leq q_i \leq {q_i}^{max} \quad \forall i \in [1,n]$

Now, the elementary servoing task is fully specified in continuous-time when all functions, models and parameters involved in equation (5) are defined.

3.2 Implementation Issues

The passage from the above continuous-time specification to a description taking into account implementation aspects is done in a strongly structured way in the ORCCAD approach. In fact, we have at this level to handle *temporal properties*, i.e. discretization of the time, durations of computations, communication and synchronization between the processes which implement every control operation. This is done by defining the basic entity called *Module-Task*.

A Module-Task (called MT in the following) is a real-time task used to implement an elementary process of the control law.

An example of the decomposition of a Robot-Task in Module-Tasks is given in section 7.1.2.

The MTs will be distributed over a multiprocessor target architecture to take benefit from increased computing power due to parallelism: in order to improve programming modularity, these MTs will therefore communicate using message passing and typed ports. Moreover, in order to make easy the automatic code generation from the graphical HMI, the structure of the MTs is the following (for a periodic task):

```
Initialization code;
   while(1){
      Reading all input ports;
      Data processing code;
      Writing all output ports;}
```

Such a structure clearly separate calculations, related to control algorithms issues, and communications, related to implementation issues and calls to the operating system.

When building the communication graph, the I/O *ports* are the only visible parts of the MTs. Each MT owns one input port for each required data and one output port for each produced data. The only operations that a MT may perform on its ports are "Send" or "Receive". The MT itself does not worry about either the name of the distant port or the semantics of the communication protocol. Implementation related properties, like the name of the connected task and the type of synchronization to be used on the pair of ports are given by the designer of the RT. This allows to reuse the MTs as objects in different RT schemes without needing any modification of their internal data structure.

3.2.1 Message passing and synchronization

In general-purpose computers and networks systems, communication protocols emphasize message ordering and large packets of data integrity, using large buffers and recovery procedures to avoid loss of data.

On the contrary, in a closed-loop control system where most of the tasks are periodic, the data beeing exchanged by the control tasks describe the state of the system: thus, we may consider that the best data to be read for control purpose is the last produced. Moreover, it is often more efficient to occasionally loose data and to wait a while for the next one, or to reuse the last available data, than to use a long recovery processe. Obviously, some messages like exception signals must be sent using safe protocols.

Often, in a rather complex structure like a RT, we may found several paths from measurement inputs to control outputs: these loops may run at different sampling periods. For example, the data related to the feedback part of the control law must be updated faster than the ones of the feedforward parts. It has been shown also that the performances of the control are influenced by the more or less tightly coupling of the cooperative tasks, according to their respective durations [14, 2].

We therefore provide the RT designer with a set of 8 communication and synchronization mechanisms to be run by the pairs of MT ports, providing the following services (P means producer and C means Consumer):

- P/asyn-C/asyn: Each task is running freely, and reads or writes on the ports when necessary. It is never blocked by the communication, thus enforcing a maximum parallelism.
- P/syn-C/syn: It is the classical "rendez-vous" for tightly coupled tasks. The first task to reach the rendez-vous is blocked until the second one is ready.
- P/asyn-C/syn: The producer is running freely. The consumer is blocked until the next data production except when a new data has been produced since the last consumption.
- P/syn-C/asyn: A symmetric protocol: the producer is blocked until a new reading if there was no pending demand, the consumer is running freely.
- P/asyn-C/synF: The producer is running freely, the consumer is always pending until a new data production.
- P/synF-C/asyn: The producer is synchronized with a new demand while the consumer is running freely.
- P/asyn-C/synD: The producer is requested to send data by the consumer.
- P/asyn-C/synDF: The producer sends its next producted data upon request of the consumer.

Note that, from a functional point of view, data transfers between MTs are unidirectional, from producer ports towards consumer ports. Nevertheless, the protocols used to actually perform the communications need bidirectional channels.

These protocols have been designed using distributing private events and encoded using the synchronous language ESTEREL [31], which is presented in section 3.4.4. They use the services of SIMPARC (see section 6) during the first simulation phase of an ORCCAD design process. For the last implementation steps of the RTs, they use the services of the real time operating system.

3.3 The Related Data Structure: an Object-Oriented Model

The object-oriented approach was selected to model Robot-Tasks for the following reasons:

- the set of all possible choices of models and the set of all task functions to be reasonably proposed lead to a finite but large number of possibilities;
- the design of a control law for a given application is not a trivial operation. The data structure should use the easiest design methodology. This is done through the use of a dedicated Human-Machine Interface (HMI), which therefore needs to insure the coherence between both aspects;
- because of the complexity of the general control scheme, any modification should be done without disturbing the coherence of the overall scheme;
- most functional components of the system may be easily described by tree models with natural inheritance properties.

3.3.1 The Classes

The ORCCAD system presently handles the following classes:

- four classes related to the control scheme (5) itself: task functions, trajectory generation, models, controls;
- two classes related to the physical entities involved by the system: physical resources, components (i.e. tools and parts);
- two classes related to the logical (event-based) behavior of the Robot-Task: observers of the elementary servoing tasks, automaton of the robot task.

As an example, figure 2 presents the class hierarchy of the class "task-function". Appendix 1 gives the associated expressions of considered task-functions. The observers and the automaton will be described later. All other classes are presented in [23].

3.3.2 Objects and Graphs

As detailed in the next section, a terminal leave in any of the class hierarchies is a single object with two kinds of attributes: the attributes relative to the *continuous-time specification* and the attributes associated with the *temporal properties*. When considering only the first ones, the object is partly instantiated, and called a functional object (*F-object*). When both types of attributes are considered, the instantiated object is called a Module-Task object (*MT-object*). We may now define two important representations of a Robot-Task:

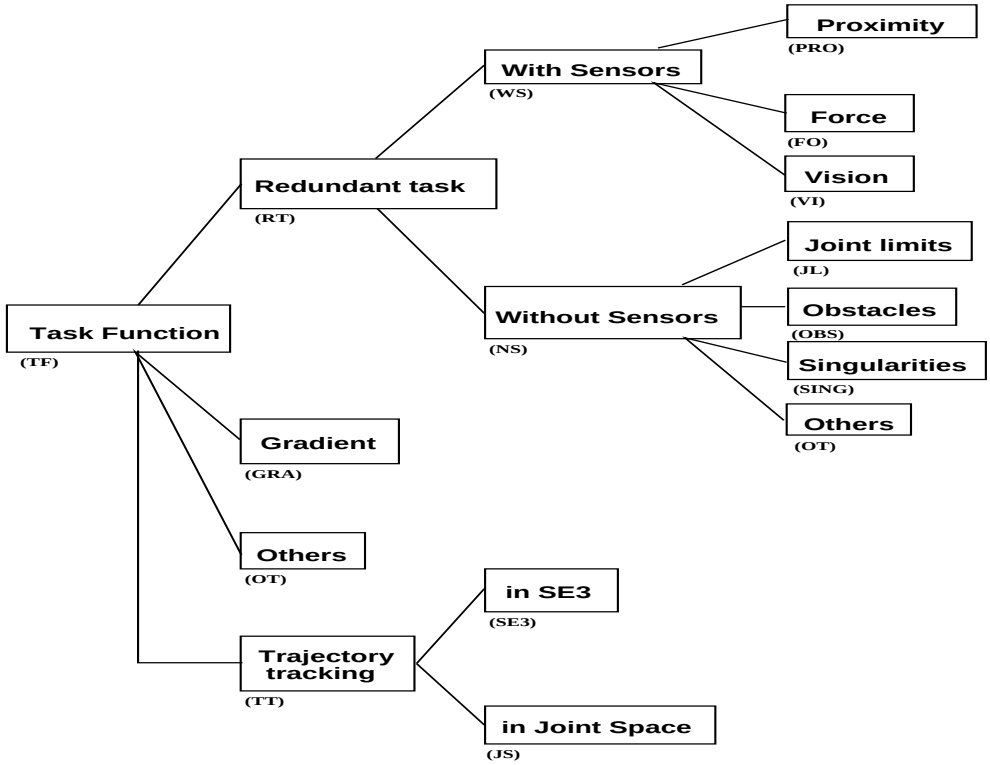

Figure 2: Hierarchy of the class Task-Function

• *The* **functional graph** *of a Robot-Task is an oriented graph. Its nodes are the F-objects instantiated for the Robot-Task, and its edges correspond to the existence of formal data transfers between F-objects.*

• *The graph of time-constrained specification (***TCS graph***) of a Robot-Task is an oriented graph. Its nodes are the MT-objects instantiated for the Robot-Task, and each of his edge corresponds to the transfer of a variable from a Module-Task to another one.*

The functional graph is close to the classical representation called *block-diagram* in automatic control. The TCS graph allows to visualize immediately the temporal properties associated to each MT by simply "clicking" into a port. Figure 5 gives an example of the TCS graphs attributes. The graphs, which may be easily edited by the user through the HMI, provides the user with a complete and synthetic representation of a Robot-Task.

3.3.3 Attributes and Methods

The objects in a Robot-Task include all information necessary to the generation of the functional and TCS graphs. Therefore, two sets of attributes are defined for every object, briefly:

1. the functional set
   - Type 1.1: name and constant non-temporal parameters of the MT
   - Type 1.2: names of the MTs to be connected
2. the TCS set
   - Type 2.1: period and duration of the MT
   - Type 2.2: communication issues between MTs, i.e. port characteristics (name, type, synchronization, name and type of exchanged variables, etc...)

- Type 2.3: localization of associated codes (initialization, computation) and name of the dedicated processor.

Finally, three sets of methods operate on these attributes, concerning respectively:

- the test of coherence between parameters (for example, dim(e)=dim(q) in equation 5);
- the automatic generation of the MT's C++ code;
- the generation of data required by the simulation system.

3.4 The Event-based Behavior

In a way, a Robot-Task is atomic for the *application* designer. However it follows an internal sequencing which the designer has not to see in normal (failure-free) circumstances. Nevertheless the RT has to exchange a minimum set of information with other RTs, in order to synchronize and/or condition their activation. In the ORCCAD framework these two aspects are considered in a single way. Thus the RTs can be considered as reactive systems and can be programmed using the synchrony assumption ([26]): signals are emitted, without any absolute reference time, from and to an automaton which specifies the Robot-Task behavior. This automaton, called RTA, is encoded using the synchronous language ESTEREL ([8]).

3.4.1 Event Generation

Signals are emitted by objects from the "observers" class. Formally, an observer is:

- *as an F-object, an object belonging to the single class allowed to communicate with the RTA class;*
- *as an MT-object, any Module-Task provided with at least one output port handling an "event-type" variable.*

Note that, due to this definition, some objects may also become observers (i.e. belong to two different classes by multi-instantiation) by adding the adequate port.

Several kinds of observers are defined in the class hierarchy. Without going into details, let us give some examples: one may monitor the task function, in order to verify that the error remains small, the occurrence of singularities (in the task-jacobian or issued from the kinematics), the approach of a joint or actuator limit, the evolution of some selected variables, the output of an external sensor (part presence, obstacle detection, DC motor overheating...), etc. From the real-time implementation point of view, a Module-Task "observer" may be either periodic or activated under interruption. All the generated exceptions are handled by the RTA described in section 3.4.3.

3.4.2 Signal Handling

In ORCCAD, *all* signals and associated processing must belong to well-defined categories. This is a way to cover a wide range of situations while allowing the automatic generation of the automaton code.

Signals. We distinguish:

- *the pre-conditions.* Their occurrence is required for starting the servoing task. They may be:

  - pure synchronization signals: flags, signals associated with a rendez-vous. Their explicit presence or absence in a RT depends on the way we chose to specify an application (see section 4).
  - signals related to the environment (also called measurement pre-conditions). An information issued from the environment is here required, usually through a sensor (part presence, physical resource availability...). Let us mention that a watchdog is generally activated on measurement pre-conditions.

- *the exceptions.* They are exclusively emitted by observers in case of failure detection (see section 3.4.1).

- *the post-conditions.* Like pre-conditions they are of two kinds:

  - logical synchronization signals emitted by the RTA itself. For example, "happy end" is emitted at time T if no exception occurred;
  - signals related to the environment, issued from a sensor (example: a force sensor monitors the arrival to contact of the desired part).

Processing. We do not present treatments associated with pre- and post-conditions since they are quite simple. The exception processing is more specific of ORCCAD and is structured as follows:

- *type 1 processing*: the reaction to the received exception signal is limited to the modification of the value of at least one parameter in MT-objects. For example, when coming near to a task-jacobian singularity, the regularization parameter λ (see appendix 1) is progressively set from 0 to 1. Or, if the error norm $\|e\|$ does not become small, the control gain k may be increased upto a maximal authorized value.

- *type 2 processing*: the exception requires the activation of a new Robot-Task. The current one is therefore killed. When the ending is correct, the nominal post-conditions are fulfilled. Otherwise, a specific signal is emitted towards the application, which *knows* the recovering process to activate. For example, if the tracked trajectory makes the robot going close to a joint limit, a reconfiguration Robot-Task is started, then the previous one may be activated again.

- *type 3 processing*: the exception is considered as fatal. Then, the robot task and the application are immediately stopped. This occurs for instance when the error norm $\|e\|$ becomes quickly too large, allowing to infer an actuator failure or a collision.

3.4.3 Specification of the Robot Task Automaton

The logical behavior of the Robot-Task may be easily specified using an intuitive syntax. Owing to the previously given types of events and treatments, three phases are defined:

- **Phase 1***:*

  $<$ *awaiting synchronization pre-condition 1* $>$

  $\|$

  . . .

  $<$ *awaiting synchronization pre-condition* n_1 $>$

  ;

  $<$ *activation of the MTs " measurement pre-condition"* $>$

- **Phase 2**:

  $<$ *awaiting measurement pre-condition 1* $>$

  $\|$

  $\ldots$

  $<$ *awaiting measurement pre-condition* n_2 $>$

  ;

  $<$ *activation of all the MTs required for the execution of the servoing task* $>$

- **Phase 3**:

  trap T2 in

  trap T1 in

  trap HAPPY-END in
  $<$ *awaiting type-1 exception 1 and processing* $>$
  $\|$
  $\ldots$
  $<$ *awaiting type-1 exception* n_3 *and processing* $>$
  $\|$
  $<$ *awaiting type-2 exception 1 and processing* $>$
  $\|$
  $\ldots$
  $<$ *awaiting type-2 exception* n_4 *and processing* $>$
  $\|$
  $<$ *awaiting type-3 exception 1 and processing* $>$
  $\|$
  $\ldots$
  $<$ *awaiting type-3 exception* n_5 *and processing* $>$
  $\|$
  [
  $<$ *awaiting measurement post-condition 1* $>$
  $\|$
  $\ldots$
  $<$ *awaiting measurement post-condition* n_6 $>$
  ]
  ;
  exit HAPPY-END
  $<$ *watching T: exit HAPPY-END* $>$
  handle T1: $<$ *kill task and emit signal S2* $>$

  handle T2: $<$ *kill application* $>$

  handle HAPPY-END:

  $<$ *emit synchronization post-condition 1* $>$
  $\|$
  $\ldots$
  $<$ *emit synchronization post-condition* n_6 $>$

3.4.4 Implementation of the Robot Task Automaton

The ESTEREL Language The previous specifications are encoded using ESTEREL ([24, 7]). Before describing implementation issues, let us recall its basic features. It is a high level, imperative language especially designed for writing reactive programs, i.e. communicating programs the behavior of which depends on the ordering of inputs and outputs events. An important characteristic of reactive systems is their intrinsic *determinism*: a reactive system determines a sequence of output signals from a sequence of input signals in an unique way. Classical programming tools are not well-suited to reactive systems programming: automata-based systems lack high-level parallel programming primitives while asynchronous languages do not respect the intrinsic determinism of reactive systems.

The main ESTEREL feature is *synchrony*: the synchrony hypothesis assumes that the transitions of the control system are atomic.

In ESTEREL, parallel processes communicate without delay by signal broadcasting. The modular structure of ESTEREL also allows a hierarchical programming. Two operators are essential: the "||" operator leads to an explicit parallelism. The branches of a 'parallel' statement start simultaneously (a 'parallel' terminates synchronously with the last termination of its branches); when using a ";" the programmers express a *sequence*, which means that the second statement of a sequence is performed exactly when the first statement terminates.

An ESTEREL program communicates with its environment via *signals*. They are used both as inputs and outputs. They are assumed to be *broadcasted* instantaneously among processes: signals are received by the different parts of the program within the same time. Two kinds of information are broadcasted: *values* that are permanent, and *signal tops* that are intermittent. The former are available in expressions, the latest act as *control information* to be handled by ESTEREL control structures. The occurrence of input signals defines the execution time of the program. Out of these occurrences, the program has no activity. Time is handled in a rigorous way since the 'absolute' time doesn't exist in ESTEREL, any signal being considered as an independent 'time unit'. Time manipulation primitives (for example a watchdog) can be uniformly used for all signals. This concept is the one of *multiform time*.

Local signals can be used for communication and synchronization purposes with other submodules. All signals are treated as messages, regardless of their hardware or software origin. Broadcasting simplifies process communication and improves modularity. As a programming language, ESTEREL is mathematically well-defined. Its implementation is based on a rigorous semantics ([25]). Finally, let us emphasize that handling asynchronous tasks using the **exec** statement will be extensively used in our controller structure ([34]).

The compiler transforms an ESTEREL program into a *finite determinist automaton*. Process scheduling and communication are performed at the compiling stage. The translation of programs to automata has a major advantage: it authorizes *automatic proofs* of the resulting automata to be performed: a property of the automaton that can be proved is associated to a precise property of the program. A verification system for parallel and communicating processes, the AUTO ([39]) system, may be used to prove that the behavior of the reactive system is the expected one. AUTO and AUTOGRAPH ([35]), a graphical tool which draws the generated automaton, use experiment verification principles such as the quotient of an automaton by an observational criterion. Checked properties may therefore be infirmed or confirmed. After the allowed verifications, one can be sure of the logical correctness of the program.

A detailed example of the use of ESTEREL in robotics may be found in [15].

Implementation The code of the Robot-Task automaton is distributed in five modules: awaiting of a measurement pre-condition (1), awaiting of a type-1 exception (2), awaiting of a type-2 or type-3 exception (3), awaiting of a measurement post-condition (4), and a main

module (5), which properly handles all the other ones. As an example, we give below the ESTEREL code for an example of module (4) awaiting the measurement post-condition POST_MES and protected by a watchdog triggered by WAIT_TIME.

```
module POST_MES :

input
   POST_MES,
   WAIT_TIME ;

output
   TIME_PASSED ;
[
   do
      await POST_MES ;
    watching immediate WAIT_TIME
      timeout emit TIME_PASSED ;
    end
]
```

At the implementation level, the previous code is accompanied by a *"main code"*, the evolution engine of the automaton, and a "*data handling interface code*" which realizes the interface with the environment. The current "event", in the sense of synchrony, is generated at this level from received signals.

Automatic Code Generation In order to remain coherent with the whole ORCCAD philosophy, it is clear that the control designer should not have to write any ESTEREL code. Moreover, ESTEREL is an imperative language, sensitive to the programming style, which requires to acquire a specific expertise. This is why the automaton code, the main code and the data handling interface code are fully automatically generated in the ORCCAD system. The considered types of events and processings allow to derive a systematic and reliable approach in the writing of automatic generation algorithms.

Therefore, to specify the event-based behavior of the Robot-Task, the user has only to instantiate the MT-object "RTA" through the HMI, by giving values to all the required attributes. For example, when defining an input port, the window of figure 3 appears and has to be fulfilled.

To point out the efficiency of the system, let us indicate that a RTA specification using one measurement pre-condition, one exception of type-1, one exception of type-2 and one measurement post-condition leads to a 3 states and 16 transitions automaton (figure 4). Writing such an automaton directly would certainly have been less reliable and more tedious. Moreover, any modification is immediately and simply taken into account by generating a new automaton.

4 The Application Level

The application level is the end users' one. It relies on a set of parametrized Robot-Tasks, the logical and temporal organization of which defines its final behavior. From the end user's point of view, this level should remain far from the implementation issues. On the contrary, it has to cope with the user's concerns. In order to facilitate the communication with this user, these concerns have to be expressed using techniques, specific terminologies or description methods issued from the *application domain* and necessarily different from a domain to another. Obviously, the ORCCAD system cannot handle by itself a set of possible application areas large

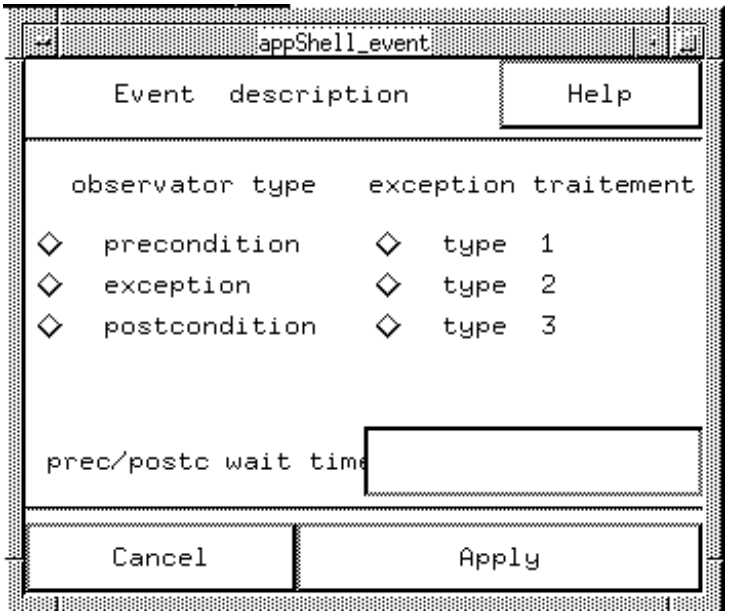

Figure 3: HMI panel for RTA specification

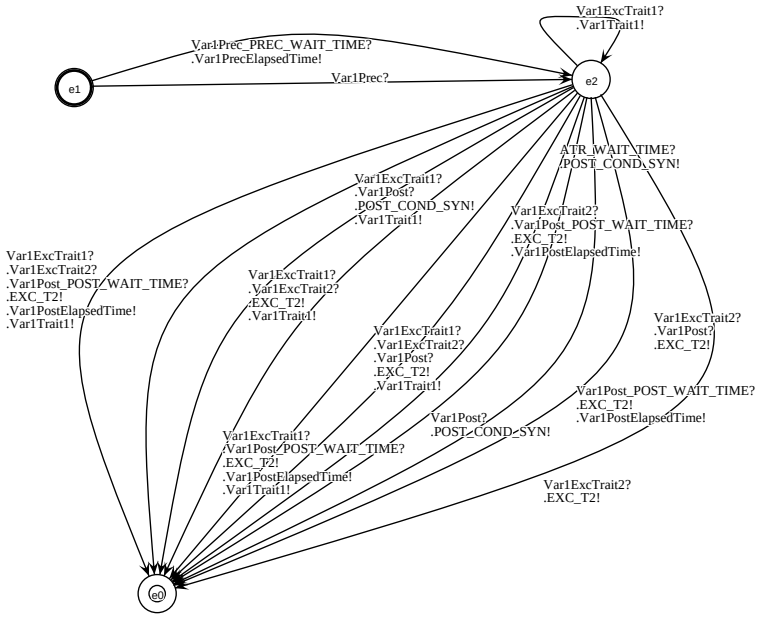

Figure 4: A Robot-Task Automaton

enough: the mission of a Mars Rover will never be described in the same way as the fitting of a windshield in automobile industry. Moreover, the application level would ideally have to be connected with various high level tools like planning or intelligent decision systems which for now should lie out of ORCCAD.

Nevertheless, and whatever the application is, we may reasonably consider that there exists an *invariant* representation scheme of an application somewhere when moving from the application to the implementation level. This is why the application features of ORCCAD are limited to such an *intermediary level*, aimed to describe in a precise way through an automaton the logical and temporal dependencies between the involved Robot-Tasks. This forms in a way the "skeleton" of any application.

Again, in order to keep the coherence and to allow the dialogue with Robot-Task Automata, the intermediary level consists in an ESTEREL program. Just as ESTEREL code is automatically generated in RTs, it may be considered that the ESTEREL application program is the output of an higher level user-oriented language or interface to be defined according to the application domain. Now, from the implementation point of view, an application is made of

- a set of Module-Tasks representing the cooperating subsystems, modeled by the MT-objects of type "components",
- the Module-Task representing the application automaton,
- and the sets of Module-Tasks who implement the elementary Robot-Tasks contributing to the application.

Let us now sketch connections between RTs and the application automaton. The characterization of logical and temporal dependencies inside an application uses two types of information: the information issued from the *environment*, and the information issued from the *execution of the Robot-Tasks*.

This information may be interpreted in different places in the general application scheme according to the way the temporal arrangement of the Robot-Tasks is established. Each choice involves a different set of connections between the application's Module-Tasks and devotes different roles to the application automaton.

Case 1: For example, suppose that two Robot-Tasks have to be executed in sequence. The behavior of a Robot-Task, as defined in 3.4.3, ensures that the transfer of dedicated information from the first Robot-Task to the second one results in that the Robot-Tasks will be executed according to the predefined plan. It suffices that this information corresponds to a post-condition of synchronization of the first Robot-Task and verifies a pre-condition of synchronization of the second one; in this way it will be executed immediately after termination of the first one. If the beginning of the second Robot-Task also depends on an information coming from the external environment, a supplementary pre-condition of synchronization in its "RTA" and a connection with the corresponding Module-Task of type "component" in charge of verifying this condition have to be added. The Robot-Task is executed as soon as the transferred information verifies the corresponding pre-condition of synchronization. Using this implementation the application automaton is in charge to activate all the Module-Tasks of the application. All the Robot-Tasks are active from the beginning of the application's execution. Only those for which synchronization pre-conditions are satisfied are executed. The application automaton is permanently waiting for two types of information from the "RTAs", i.e. an exception of type 2 or of type 3 is encountered. In the first case the application automaton aborts the particular Robot-Task and activates a sequence of others in order to recover the error, in the second one the application is aborted.

Case 2: A different implementation consists of gathering all information issued from the environment and the Robot-Tasks in the application automaton. We recall that the Robot-Task

can be parametrized by two sets of parameters; the input one used for executing a Robot-Task in different contexts and the output one used to inform the application about the termination state of the RT. Three different situations may occur: the Robot-Task is well terminated, or an exception of type 2 or of type 3 is encountered. Using the *exec* statement ([34]) of Esterel we have the possibility to activate Robot-Tasks and collect in local variables the way the Robot-Task is terminated. In this way, the internal Robot-Task information can be made available in the application automaton. Some Module-Tasks and signals can also be used directly at the application level to implement an information flow between the application automaton and the environment. The application automaton specify now the application skeleton and activates the Robot-Tasks accordingly to the set of satisfied conditions. The "RTAs" do not wait for pre-conditions of synchronization. The exceptions handling follows the same principles as described earlier. This second solution is used in the example of section 7.2.

To conclude with, let us mention that a first version of an application programming language is proposed in [17] and [18]. Aimed at the specification of plans of actions at the level of sets of RTs, this language is provided with imperative control structures and is defined in terms of temporal logic, in relation with artificial intelligence planning. A compiler translates plans written in the application language into ESTEREL code corresponding to their temporal definition. In this way the end user of ORCCAD is offered a language enabling him to express a sequencing of Robot-Tasks with no need to investigate the details of the signal exchanges, dialogues and implementation issues for each particular arrangement of tasks.

5 Human-Machine Interface

In all CAD systems, the efficiency of the Human-Machine Interface conditions the overall performance of the design system. In ORCCAD, the HMI is also a key tool, presently oriented towards users with an automatic control background. It aims to assist such an user in the process of Robot-Task creation: specification of the control scheme and of the local behavior, and addition of temporal features. Owing to its connection with the SIMPARC simulator described in section 6, it also allows to test a Robot-Task and therefore to easily adjust its characteristics.

The components of the system are a main menu bar, one or more windows to generate the graphical representation of the specified objects and a set of interfaces to get their characteristics. The editing functions are executed after selection from a menu in the menu bar and the pointing with the mouse on a specific desired location where it makes sense to execute them, or by entering textual information in dedicated interfaces (figure 5).

The basis of the HMI is the object-oriented structure presented in section 3.3. It is implemented in C++ under Motif/X Windows. Several hierarchies of classes are implemented and proposed to the user via a popup menu in the selected window. A specific choice makes dedicated interfaces to appear, where the user, guided by the system, enters its specifications and instantiates the leaves of the hierarchies called *F-objects*, defined formally in section 3.3.2.

The *F-objects* are graphically represented by rectangles where the existence of an interconnecting line points out that there is a data transfer from one *F-object* to another. Two *F-objects* can be connected if a set of tests of coherence is satisfied, validating in that way the end user specifications. The control law and the physical behavior of the Robot-tasks are so specified in a continuous-time model.

The specification of the ports characteristics, as well as the temporal attributes of the leaves of the hierarchies, transform the continuous-time model in the TCS model. Graphically, ports are represented by circles, and connections by lines, generating a scheme like figure 6, called *TCS-graph*, and formally defined in section 3.3.2. The tests of coherence at this stage mainly concern the verification of the conditions to be satisfied for connecting two distant ports (different types of ports (I/O), same type of synchronization, same type of transferred data, ...).

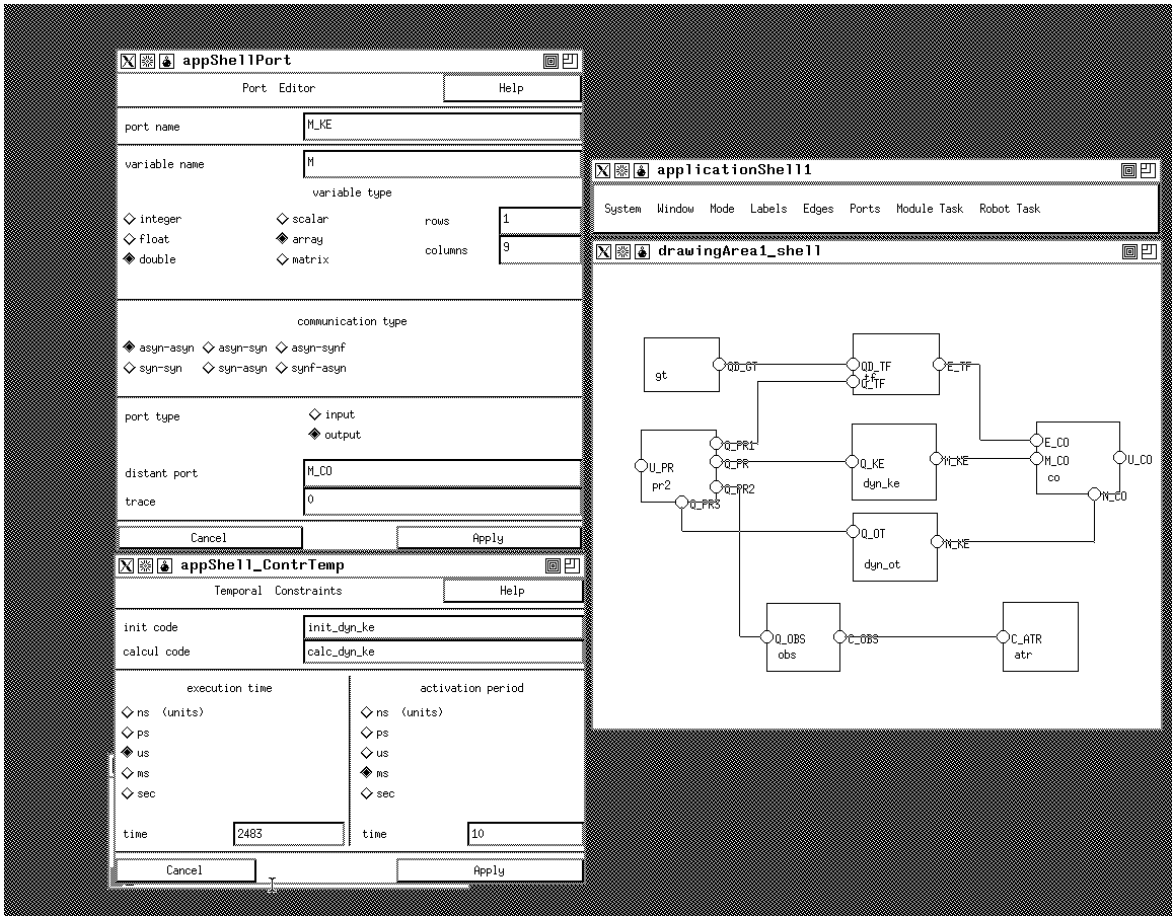

Figure 5: HMI panel for TCS specification

Once the algorithmic and temporal attributes, as well as the physical behavior of the system to control have been defined, the user can proceed to a simulation, using SIMPARC. The HMI translates the *MT-objects* to Module-Task devices of SIMPARC, generates automatically the data processing code for every *MT_object* and creates the necessary environment to proceed to the simulation.

Note that, for achieving the hardware simulation and the downloading of the code generated, new attributes must be added to the *MT-objects*. These attributes are related to the processor on which the Module-Task will run and to the operating system calls used to instantiate and activate real time tasks.

6 Simulation Software for ORCCAD

The interest of simulation tools in the process of CAD of robot controllers was mentioned in section 2.2. Once a RT has been specified with ORCCAD, two simulation phases are required to validate the choices done by the designer: the first one deals with the TCS properties evaluation while the second is very close to implementation and is the last step before downloading.

6.1 The Basic SIMPARC Simulator

Control systems, and more especially robotics system, are hybrid systems from the time point of view. The controlled system belongs to the physical world, and can generally be described by a set of differential or partial derivative equations in continuous time.

On the other side, the controller works basically in the frame of discrete time. For this part of the robotics system, time is usually modeled using events ordering, delays, time stamps or synchronization points. The controller itself is made of a large set of concurrent and communicating sub-processes.

Although simulation tools exist for both continuous and discrete time systems, it would be difficult and unefficient to connect these two kinds of tools in order to achieve realistic simulations of a robotics system.

SIMPARC (Simulator for Multiprocessors Advanced Robot Controllers) ([3, 4]) was designed from scratch to handle these two aspects within a single simulation system. This simulation tool actually runs the user's control programs on the simulated target architecture and is mainly concerned with time management.

The target hardware architecture is described as a set of communicating devices. The simulation tool is based upon an event driven simulation kernel coupled to a numerical integration package. Figure 6 shows the general organization of the system. The kernel manages a set of independent processes associated to the devices. The devices communicate when the associated process calls a function of another device. The processes generate events to the kernel by calling appropriate timing requests (computation time, data transfer). The kernel also manages the continuous time evolution of the robot equations between the occurrence of events.

The software was developed in C++, following an Object Oriented Design methodology. All the objects handled by SIMPARC may be customized using time or hardware-related parameters. The main objects of the current library are:

- The current processor device, which simulates the behavior of a general purpose processor with vectorized interrupts. It is user-programmable, since it runs the *actual* user-defined control tasks (written in C), to which information are added about their assumed execution duration.

- The continuous component, which handles the files describing the physical system. This system is described in a state model with a set of differential equations. This component

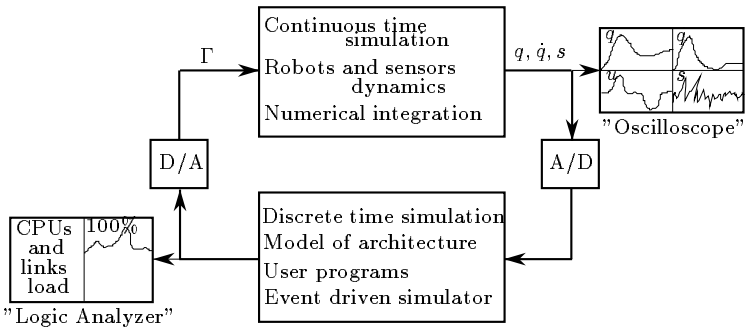

Figure 6: General Organization

calls a classical numerical integration algorithm at appropriate instants.

- The Analog to Digital and Digital to Analog components, which are used to perform communications between the continuous and discrete time worlds, where data are not encoded in the same representation.
- The Pins and Buses, which are basic components used for devices communications. The special Xbus component is used to perform address translations over multiprocessors backplanes. General purpose components like memories or real-time clocks are also needed to complete an architecture design.

Simulation results are available from the two sides of the software:

- The user may point out events of interest to be monitored and stored by SIMPARC. The analysis of the generated file allows to trace relevant variables like CPU's load or bus contention.
- Data coming from the continuous system simulation may be displayed. The analysis of tracking errors or of control amplitude may for example help to tune control loops gains.

The basic SIMPARC software allows to perform simulations which are very close to actual implementation, since it uses actual programs to be run and an accurate description of the robotics system. Thus, small efforts will be further necessary to produce code for downloading.

An example of the design of a single-processor controller is shown figure 7.

6.2 Simulation Tools for the Evaluation of Time-Constrained Specifications

Inside the general control scheme (5), the RT designer has to do many choices about the models to be used and the temporal properties assigned to the MTs.

Many questions about these choices cannot currently be solved by mathematical analysis. The control algorithm may not require to be fully computed at each sample period. It is generally possible to distinguish between the critical path (feedback path) and non critical computations (gain adjustment, feedforward). The computation rates of the different parts of the algorithm may affect the overall behavior of the robot as well as the controller design. Sometimes, simple and fastly computed models might provide better overall performances than a slower sophisticated control law. As it does not appear that general answers exist, these choices will have to be checked for every new design.

At the TCS design step, the main objects to be taken into account are MTs and ports: models of these objects have been added to the basic SIMPARC software. They are the only

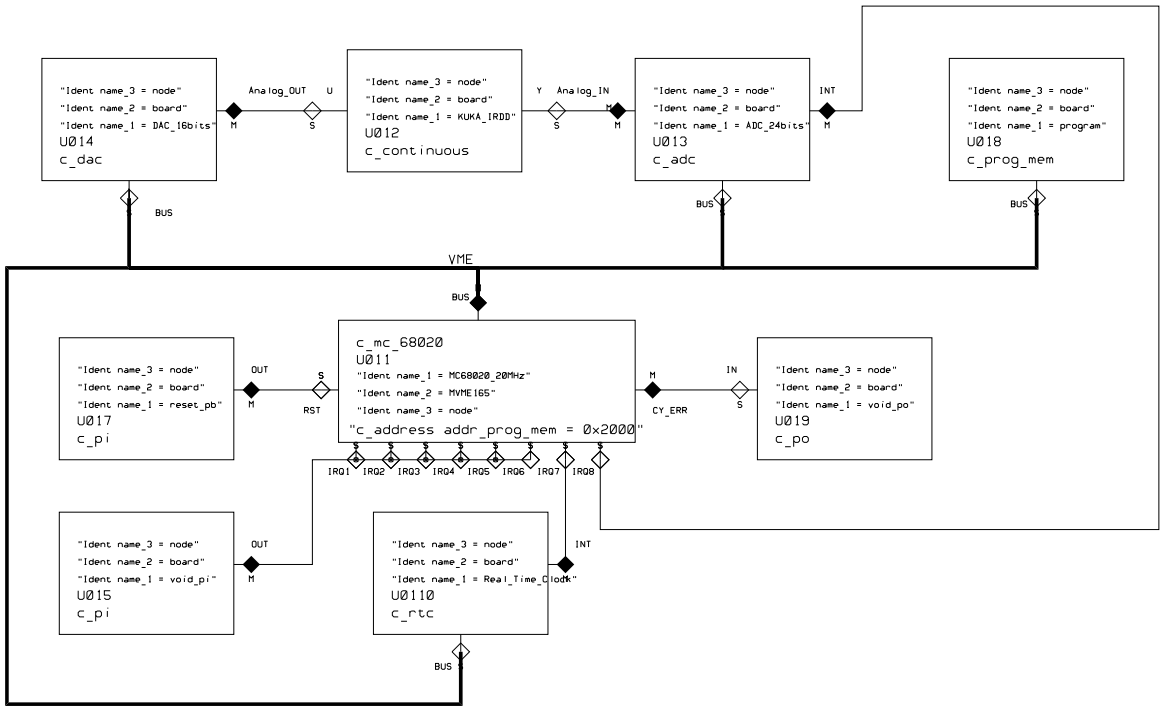

Figure 7: Mono-processor robot controller

objects handled by the TCS designer, all the basic mechanisms of the SIMPARC kernel are still used but are hidden. The set of MTs, ports, connections and temporal properties are designed using the HMI. A translator then instantiates all the C++ classes needed for simulation.

The synchronization mechanisms used by the I/O ports have been encoded with ESTEREL and call services of the SIMPARC kernel.

Quantitative results about the data flow between MTs during simulation are available. This kind of result can be used later to help a semi-automatic placement of MTs on a multiprocessors architecture, where an important criterion will be to place on the same processor MTs with strong data exchanges, in order to avoid backplane bus contentions.

7 Two Examples

We will now illustrate the Robot-Task and Application levels of ORCCAD through two examples.

7.1 TCS Simulation of a Joint-Space Control

In this first example we consider a three degrees of freedom direct-drive robot. We present successively the implementation of a PID control law and of a computed torque control.

7.1.1 Robot-Task description

The PID control law has the advantage of requiring neither the knowledge of the model structure nor the model parameters to solve in most cases with efficiency the regulation problem. The dynamic choices do be done in equation (5) are:

$$\hat{M} = I \quad \text{and} \quad \hat{N} = 0$$

The task-function of a trajectory tracking in joint space can be easily written as:

$$e(q,t) = q - q_d(t) \tag{6}$$

and therefore

$$\frac{\partial e}{\partial q}(q,t) = I \tag{7}$$

The G and D tuning matrices can be fixed as identity matrices:

$$G = I \quad \text{and} \quad D = I$$

With the previous assumptions, the general form of the control (5) to which is added an integral action becomes:

$$\Gamma = k_p e(q,t) + k_v \dot{e}(q,t) + k_i \int e(q,t)dt \tag{8}$$

In our case, the trajectory to be tracked is specified as:

$$q_d = q_0(1 + cos(\pi t)) \tag{9}$$

$$\dot{q_d} = -q_0 \pi sin(\pi t) \tag{10}$$

with the initial position: ${q_0}^T = [.5,\ .5,\ .5]$ rad.

The logical behavior associated to this RT is simply described by a "joints limits" observer. The observer will emit a signal as soon as a robot joint becomes near one of its limits, raising a type 3 exception.

7.1.2 The Control Block-Diagram

The previous control scheme was specified with the HMI and the result is shown figure 8. The names of the variables to be transferred and the kinds of communication mechanisms are indicated for every port-to-port connection. The two parameters inside the boxes represent the activation period of the MTs and their assumed execution duration.

The **PR.MAN** box is only used for simulation purpose as a model of the physical system. It handles a state representation of the robot arm under the form:

$$\dot{X} = f(X,U,t), \qquad Y = g(X,U) \tag{11}$$

which in our case takes the basic form:

$$\begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} q \\ \dot{q} \end{bmatrix} + \begin{bmatrix} 0 \\ M(q)^{-1}(N(q,\dot{q}) - \Gamma) \end{bmatrix} \tag{12}$$

to which may be added "analog" functions like current loops, sensors non-linearities and torque limits.

The user gives through the HMI the names of the C files corresponding with the chosen f and g functions.

7.1.3 The Time Constrained Specification

The MTs durations of execution are estimated for a 16 MHz mc68020 processor.

The MTs activation periods were progressively tuned to obtain satisfactory control performances.

The communication mechanisms associated to ports were chosen to avoid dead-locks.